Randomised Optimization

Travelling Salesman Problem

The Travelling Salesman Problem (TSP) is the problem of finding the shortest yet most efficient route to take given a list of specific destinations. It's an interesting problem because the time complexity of the problem increases exponentially with the number of destinations and approaches like dynamic programming are not feasible for a large number of cities . For the purpose of this experiment cities are represented by random points in a 2d plane plane

Random Hill Climbing

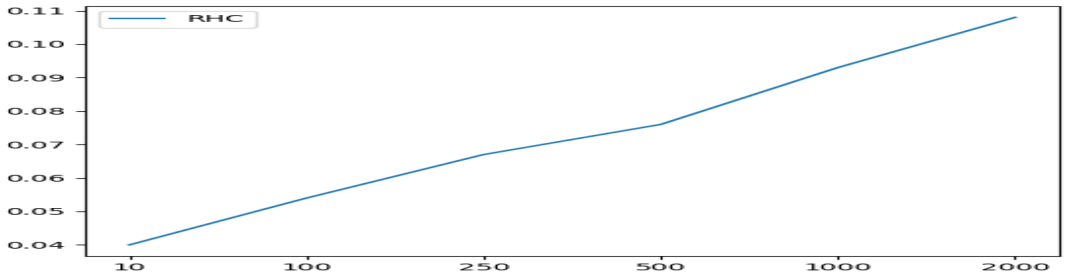

Fig 1: Fitness value vs number of restarts

Random hill climbing would start with cities in a random order and try to rearrange(eg. Swap adjacent cities or try in reverse order) the cities as to improve fitness function(minimise inverse of total distance).Fitness value increases with number of restarts because RHC is able to explore the problem space better(arrange cities in different order) and and find the optimal maxima, with fewer restarts it gets stuck at local maximas because it doesn't have enough opportunities to try out different arrangement for cities.

Simulated Annealing

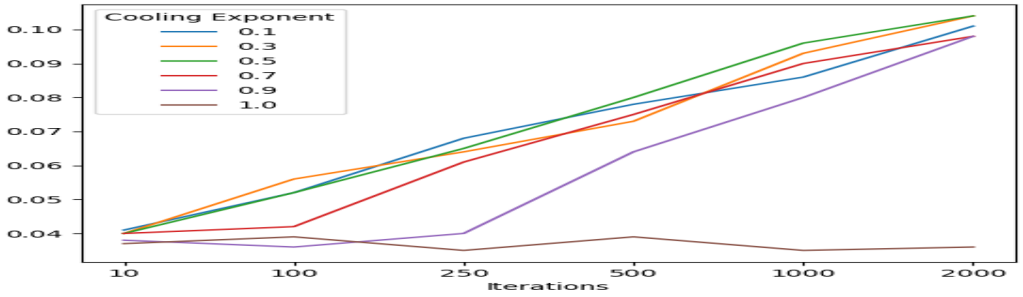

Fig 2: Fitness value vs number of Iterations for different cooling schedule

Simulated annealing is able to optimise the fitness at low values of cooling schedule , because it gives SA enough opportunities to explore suboptimal arrangement of the cities and as it cools down exploit the optimal arrangement of the cities further. It doesn't perform as good as RHC because RHC follows up a random arrangement till it reaches the local optima, SA on the other hand may jump around from different arrangements and exploit a certain arrangement only late in the iteration when it has cooled down significantly and also SA sample new points from the neighbourhood of the current point it's unlikely for SA to try radically different arrangement of cities

Genetic Algorithm

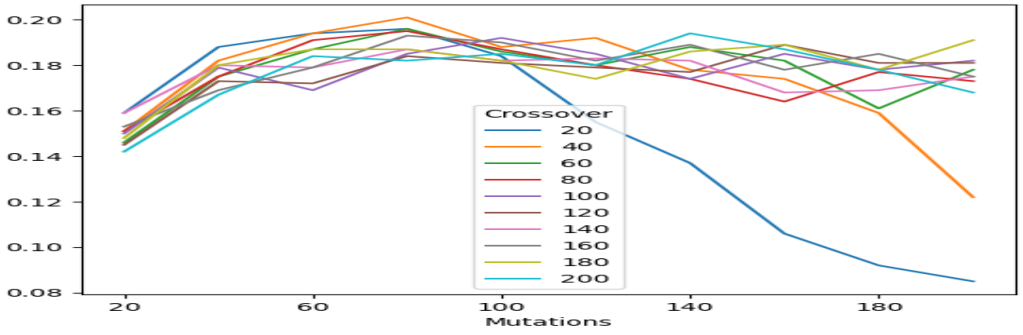

Fig 3: Fitness value vs number of Iterations for different cooling schedule

GA optimises the fitness function for crossover value of 40 and mutation count of 80.It tend to get stuck at local maxima for extreme values of crossover and mutation. This is because for high level of mutation there is an increase in diversity of the population due to which GA can explore the problem space far and wide but due to low values of crossover it is not able to exploit the problem space. Similar due to high crossover good parts of healthy population survive the next iteration due to which they are able to exploit spaces around local maxima but due to low values of mutation its not able to explore the space fully.

MIMIC

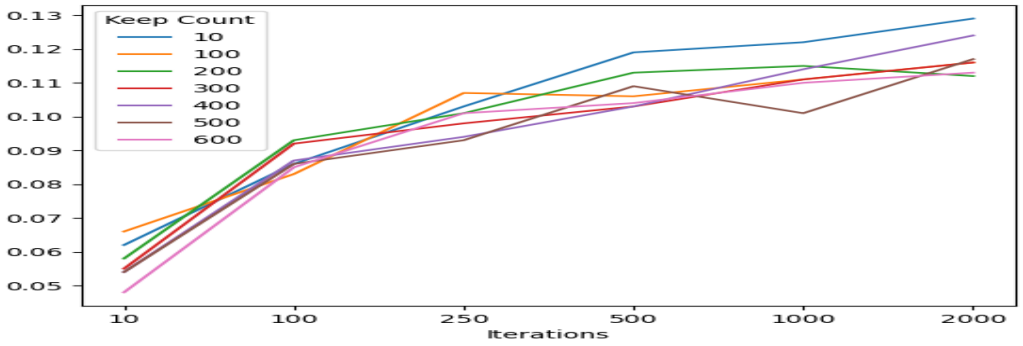

Fig 4: Fitness value vs number of Iterations for different Keep Counts

MIMIC doesn't perform as well as GA because estimated pairwise conditional probability distribution may not have been very accurate for TSP because the arrangement of each city impacts every other n-1 city. Due to which the generated sample may not have captured the structure of this problem space accurately and got stuck in some local maxima.

For fewer iterations MIMIC gets stuck in local optima for high keep count because iterations are not enough to either explore and exploit. MIMIC performs well for low keep count as the iteration increases its very likely due to TSP's problem space which doesn't have many bumps and even with low keep it doesn't miss out on global optima

Performance Comparison

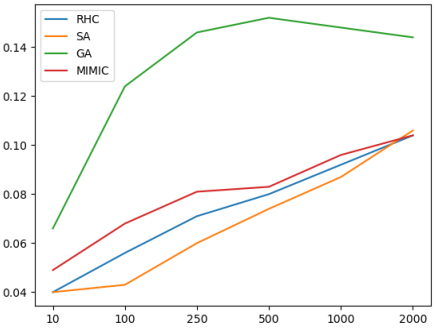

Fig 5: Fitness value vs number of Iterations for different Algorithms. .

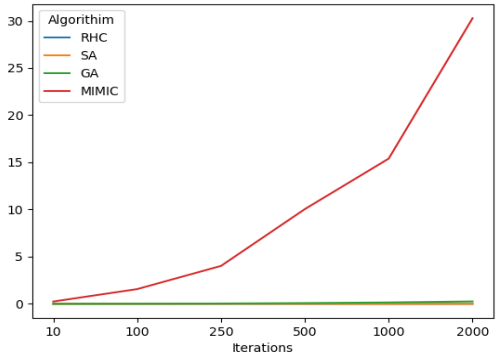

Fig 6 : Runtime for Algorithms

GA performs significantly better than other algorithms because for a TSP problem adjacent cities from parents with high fitness are carried over to the offsprings as part of the crossover without duplicate visits. These blocks of adjacent cities coupled with small mutations on optimal individuals in the population help achieve the optimal maxima. And also GA search can move around more abruptly ,due to mutations offspring may be very radically different from the parent it is very less likely to get stuck in local maxima even during later stages of iteration and since the fitness function is sum of the distance between adjacent cities GA could try out wholly different combinations of cities and mutation can help improve upon already optimal combination of cities. Mimic has higher runtime due to the fact that for each iteration it has to sample the from distribution and re estimating the distribution

Four Peaks Problem:

Four Peaks is a problem which as two local optima and two global optimas similar to the function shown below

$$z(x) = \text{Number of contiguous Zeros ending in Position 100}$$
$$o(x) = \text{Number of contiguous Ones starting in Position 1}$$

$$REWARD = \begin{cases} 100 & \text{if } o(x) > T \wedge z(x) > T \\ 0 & \text{else} \end{cases}$$

$$f(x) = MAX(o(x), z(x)) + REWARD$$

For T = 10 this function has two two local optima at o(x) =0 , z(x) = 100 and o(x) =100 , z(x) = 0 and two global optimas at o(x) = 89 , z(x) = 11 and z(x) = 89 and o(x) = 11. Its a problem with a relatively simple problem space can help compare different optimization algorithms

Random Hill Climbing

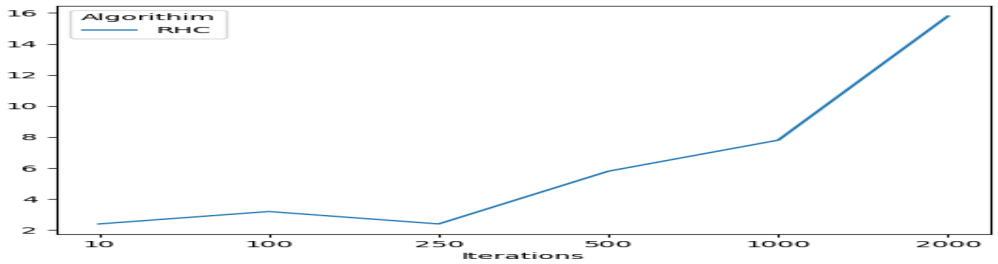

Fig 5: Fitness value vs number of Iterations

With increasing number of iteration odds of random hill climbing landing at the edges of the problem space increases where it's likely to find the optimal maxima. Four peaks problem has two suboptimal local maxima with a wide valley in between where RHC tends to get trapped due to which it performs poorly compared to other algorithm

Simulated Annealing

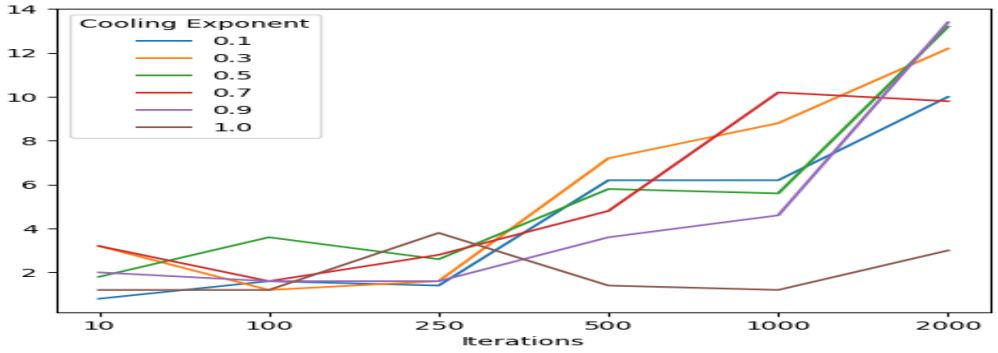

Fig 6: Fitness value vs number of Iterations for different cooling schedule

SA does not perform as well as RHC, because it gets stuck at two local optima and since it samples new points from the neighbourhood of the current annealing it is not able to explore far and wide.. At a higher cooling exponent the probability of landing at the edges which contains global maxima reduces.

Genetic Algorithm

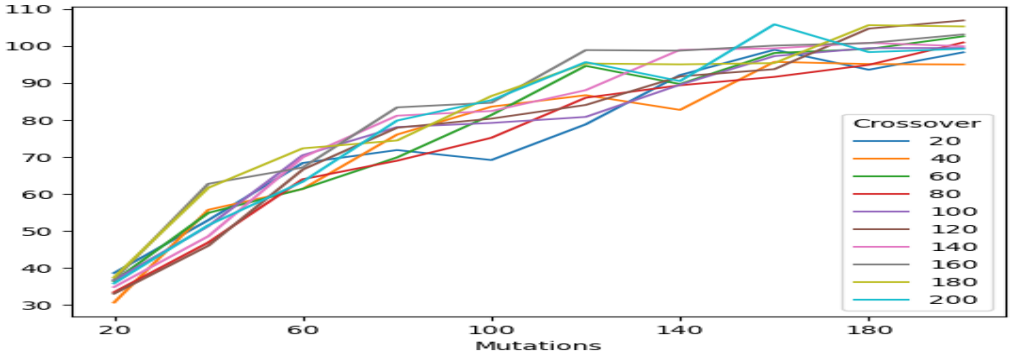

Fig 7: Fitness value vs number of Iterations for Crossover

GA performs significantly better than SA and RHC , because GA is not restricted to single bit hillclimbing(continue to increase a bit pattern for o(x) or z(x) until it reaches local optima). Crossover among bit string populations with lower fitness members may lead to bitstring patterns with higher fitness value.

MIMIC

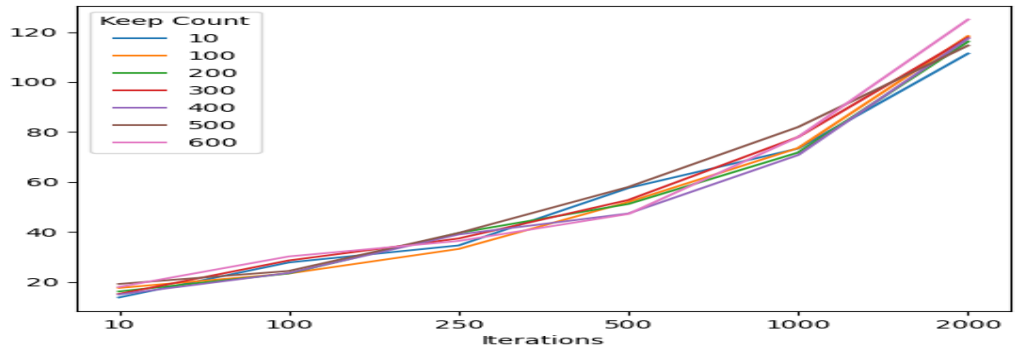

Fig 8: Fitness value vs number of Iterations for different Keep counts

Fitness function keeps increasing for all values of keep count with increasing number of iteration. It's because of the relatively simple structure of problem space

Performance comparison

As we can see from the graphs above MIMIC performs better among all the other algorithms because the pairwise conditional probability between O(x) and Z(x) as shown in the function above can be estimated accurately. As a result the samples generated can capture the relatively simple structure of the four peaks and combine information from all the four maximas. MIMIC has the highest runtime for the reason explained before

N Queens Problem

N Queens Problem is an optimization problem belonging to the class of NP-Complete Problems. The objective of N Queens Problem is to suitably place N number of Queens on an N x N chessboard in a way that there is no conflict between them due to the arrangement, that is, there is no intersection between them vertically or horizontally or diagonally. An interesting aspect about this problem is that positions of all the queens tend to affect every other queen; it would test the effectiveness of the MIMIC algorithm which creates a dependency tree for sampling. It would be difficult to create a dependency tree which is not acyclic

Random Hill Climbing

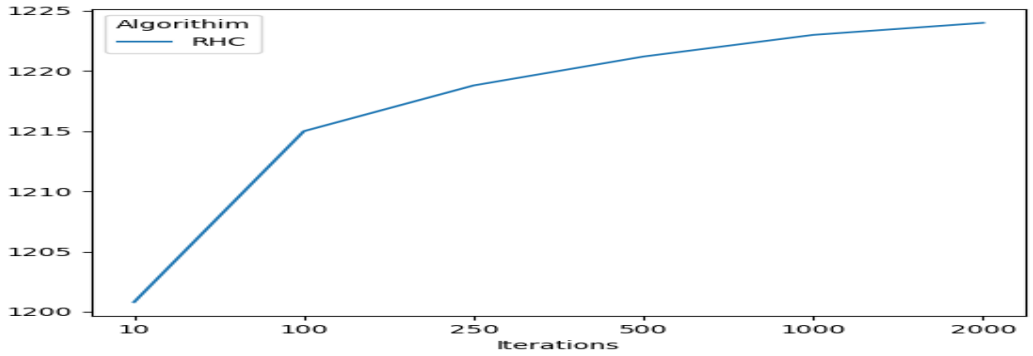

Fig 9: Fitness value vs number of Iterations

Fitness function increases with number of iteration then plateaus out , for large number of restarts it was able to explore the problem space and was able to reach ideal placement for N queens. Unlike GA which tends to carry over some optimal positions of the N queens to its offspring, RHC tends to start from a random position on every restarts its never able to build upon optimal arrangement till has come across in previous iterations

Simulated Annealing

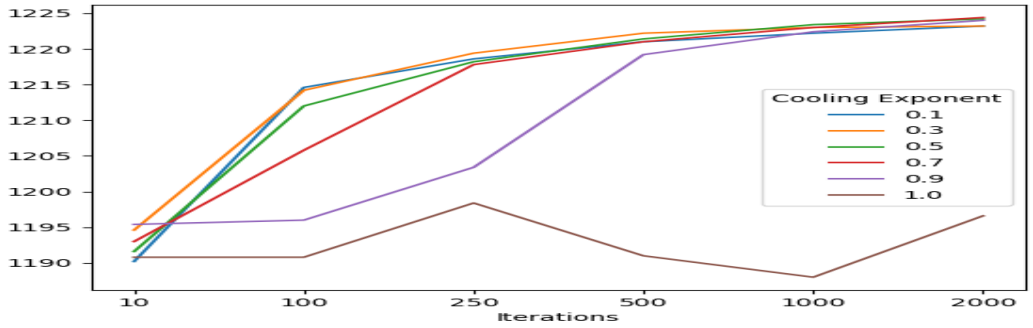

Fig 10: Fitness value vs number of Iterations for different cooling schedule

Simulated annealing performs relatively well for lower cooling exponent because its able to explore radically different arrangements of N queens and towards the end of the iteration make small adjustments to optimal N queen arrangement.

Genetic Algorithm

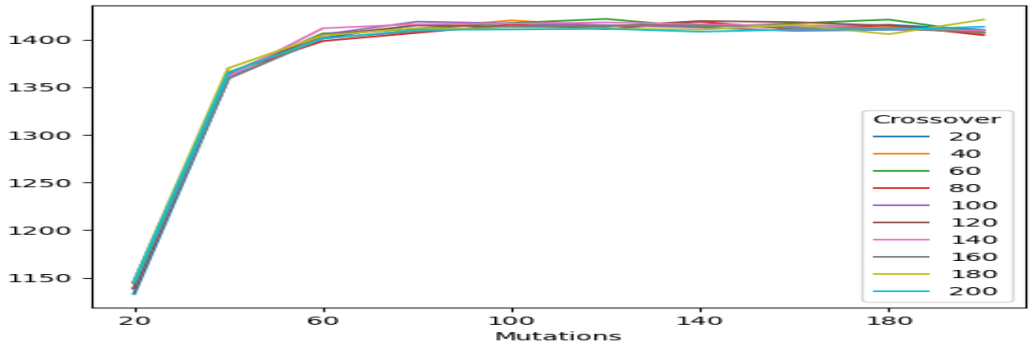

Fig 11: Fitness value vs number of Iterations for different Crossover values

GA performs better than both RHC and SA and performs better for higher counts of mutation. For N queens problem position of each queen form the chromosomes due to crossover, positions from healthy parents get passed on to offsprings and high rate of mutation can lead to exploration of more optimal position of the N queens

MIMIC

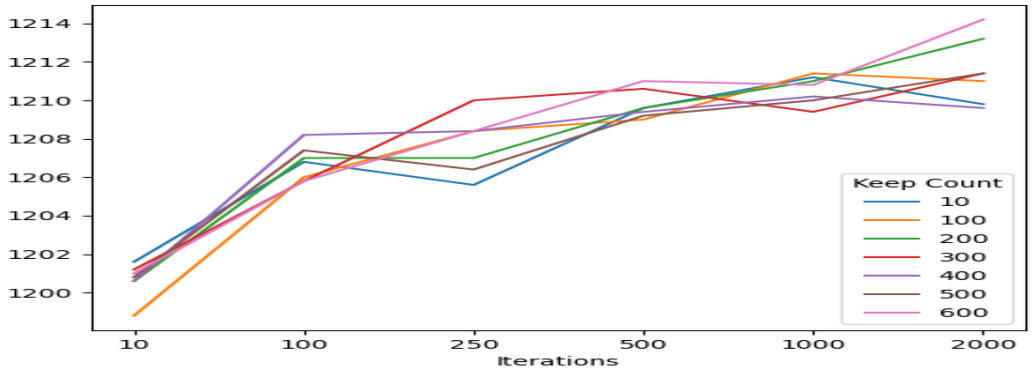

Fig 12: Fitness value vs number of Iterations for different Keep Counts

MIMIC performs the worst on N queen this is because for a N queens each queen impacts the position of the rest of the N-1 queens , the dependency among the features(position of the queens) would be more like a fully connected graph.The dependency tree where each feature has only one parent is not able to capture such a relation. Hence the distribution generated from the dependency could miss out on the optimal points in the problem space.

Performance comparison

As shown in the graphs above Genetic algorithm performs better than other algorithms because GA is able to capture optimal positions from parents and combine them into offspring using crossover and mutation enables it to consider radically different positions too, unlike RHC and SA which don't have mechanisms to capture learnings from previous iterations. MIMIC has a mechanism to capture structure of the problem space but thats dependent on estimating the pairwise conditional probability among the positions of all the N queens

Neural Network

I have used a Credit card fraud detection dataset from assignment 1. The problem here is to identify a fraudulent transaction, so that the customers are not charged for purchases they didn't make, and to discourage credit card thefts. Using algorithms of randomised optimization we would like to network weights for the optimal hypothesis. The optimal hypothesis in this is one which minimises misclassification over the test data. I employed three Randomised optimization algorithms Random Hill Climbing, Simulated annealing and Genetic algorithm. I kept the size of the network with two hidden layers [7,7] based on the grid search from the last assignment.

Random Hill Climbing

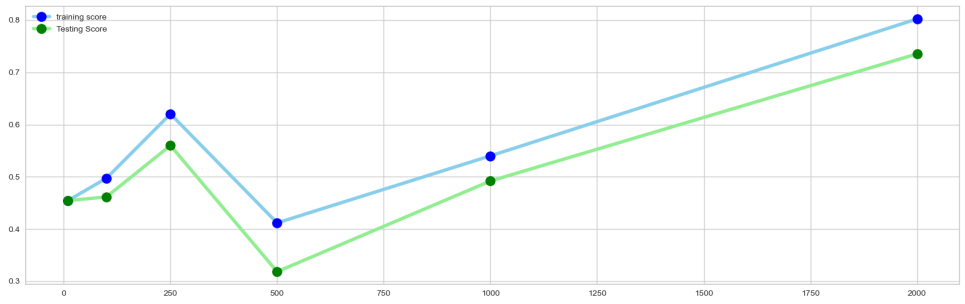

Fig 13: Accuracy vs Number of restarts

Both training and testing accuracy is low when the number of restarts is low. It increases with no. of restarts there is a sudden drop at around 500 restarts because the random state is initialised to random seed for every restart RHC got unlucky and got stuck in local minima.

Simulated Annealing

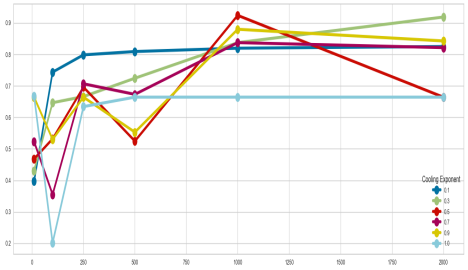

Fig 14: Training accuracy

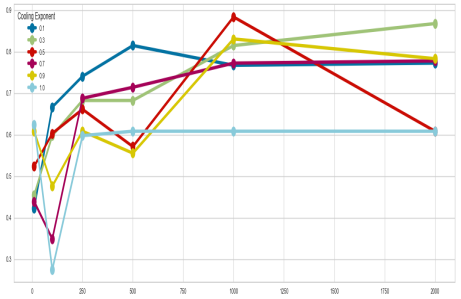

Fig 15: Testing accuracy

It performs the best for a cooling exponent of 0.5 which provides a good trade off between exploring the problems space and exploiting the neighbourhood around optimal maximas

Genetic Algorithm

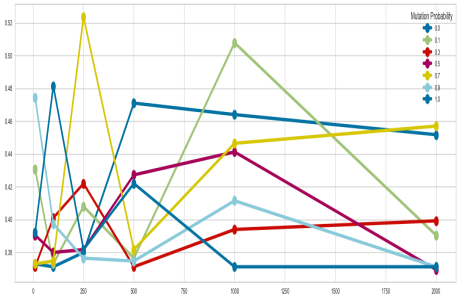

Fig 16: Training accuracy

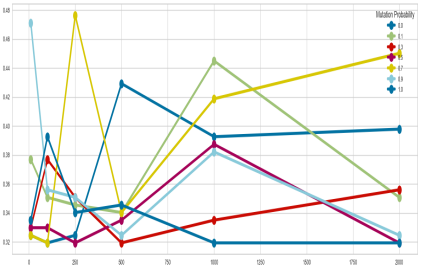

Fig 17: Testing accuracy

GA performs very poorly for both the training and testing dataset its not able to find optimum weights for the neural network , it was initialised random weights.Its very likely because the network is not complex enough to learn the optimal hypothesis

Reference

1. https://www.mdpi.com/2078-2489/10/12/390/pdf
2. https://www.ri.cmu.edu/pub_files/pub2/baluja_shumeet_1995_1/baluja_shumeet_1995_1.pdf
3. https://faculty.cc.gatech.edu/~isbell/tutorials/mimic-tutorial.pdf
4. https://www.hindawi.com/journals/cin/2017/7430125/
5. https://www.kaggle.com/janiobachmann/credit-fraud-dealing-with-imbalanced-datasets
6. https://faculty.cc.gatech.edu/~isbell/tutorials/mimic-tutorial.pdf
7. https://www.mathworks.com/help/gads/how-the-genetic-algorithm-works.html
8. https://towardsdatascience.com/evolution-of-a-salesman-a-complete-genetic-algorithm-tutorial-for-python-6fe5d2b3ca35
9. https://github.com/pushkar/ABAGAIL
10. https://faculty.cc.gatech.edu/~isbell/papers/isbell-mimic-nips-1997.pdf
11. https://chris-said.io/2016/02/28/four-pitfalls-of-hill-climbing/